

Zum tätigkeitstheoretisch orientierten Ansatz in den achtziger Jahren und heute

CHRISTIAN DAHME
dahme@hu-berlin.de

Der tätigkeitstheoretisch orientierte Ansatz in der Informatik geht insbesondere auf die Tätigkeitstheorie der Psychologie (Wygotski 1964, Leontjew 1979) zurück und versucht, die Beschreibung von sozialen Systemen und Prozessen mit Hilfe von Tätigkeiten für die Informatik zugänglich zu machen.

1 Modelle sozialer Systeme und Tätigkeitstheorie

Anfang der achtziger Jahre spielten Modelle als Voraussetzung für die Entwicklung von Software noch eine dominierende Rolle. Das war auch dann der Fall, wenn man sich Gebieten, wie dem Städtebau oder dem Gesundheitswesen zuwandte. Die Besonderheit bestand hier darin, dass es sich aus Sicht der „Angewandten Systemanalyse“ um soziale Systeme handelte. Ein soziales System wurde dabei so definiert, dass zu diesem Menschen gehören, die durch Tätigkeiten miteinander in Wechselwirkung stehen, und diese Tätigkeiten einem gewissen Ziel dienen, das für dieses System charakteristisch ist.

Es ging dabei um einen Systembegriff, in dem Tätigkeit und Ziel eine wesentliche Rolle spielten (Dahme 1997, 1987). Hierfür wurden verschiedene Tätigkeitsbegriffe herangezogen, so auch der Marxsche Begriff der Lebensweise. Von hier führte dann ein direkter Weg zur Tätigkeitstheorie der Psychologie (Wygotski 1964, Leontjew 1979).

Es gab hierbei mehrere Probleme:

- a) der Zielbegriff (was versteht man unter einem Ziel, ...),
- b) der Kooperationsbegriff (was versteht man unter Kooperation, was sollte er beschreiben und wie hängt er mit Selbstorganisation zusammen, ...),
- c) welcher Tätigkeitsbegriff ist hierfür geeignet?

Ende der achtziger Jahre wurde daher von mir der interdisziplinäre Arbeitskreis „Tätigkeitstheorie und Selbstorganisation“ ins Leben gerufen, um einerseits solche Probleme zu klären und andererseits mit Hilfe der Tätigkeitstheorie und der Selbstorganisation dem Verständnis sozialer Phänomene näher zu kommen (Dahme 1989, 1990, 1991, 1996, 1997, 1998). Die Ergebnisse waren

auch für die Softwareentwicklung im Rahmen des gegenstandsorientierten Modellansatzes hilfreich (Dahme 1993, 1995).

Darüber hinaus gab es in Skandinavien um Yrjö Engeström (1987), der eine spezielle Richtung der Tätigkeitstheorie vertrat, eine Strömung in der Informatik, die seinen Ansatz nutzte. Dieser diente insbesondere zur Bewertung von Software als Werkzeug in Arbeitsprozessen, jedoch nicht zur Konstruktion von Software. Diese Richtung spielt im Folgenden keine Rolle.

2 Der tätigkeitstheoretisch orientierte Ansatz

In enger Zusammenarbeit mit Arne Raeithel entstand in der ersten Hälfte der neunziger Jahre ausgehend von der Leontjewschen Tätigkeitstheorie (Leontjew 1979) der selbständige tätigkeitstheoretisch orientierte Ansatz (Dahme/Raeithel 1997). Im Unterschied zum gegenstandsorientierten Modellansatz, der die Bildung eines gegenständlichen Modells voraussetzt, stand hier die Tätigkeit, die gegebenenfalls in Software zu übertragen war, im Vordergrund. Dabei waren folgende Fragen von Interesse:

- Welcher Anteil einer Tätigkeit hat die Potenz zur Automatisierung und lässt sich gegebenenfalls in Software übertragen?
- Wie kommt man zu diesem Anteil?
- Wie geht man mit dem nicht formalisierten oder dem nicht formalisierbaren Anteil um, der für die adäquate Nutzung der Software stets notwendig ist?

Der tätigkeitstheoretisch orientierte Ansatz versucht solche Fragen zu klären. Nach diesem Ansatz ist der Anteil einer Tätigkeit, der sich potentiell in Software übertragen lässt, durch folgendes charakterisiert:

- „1. Er bezieht sich auf Anteile einer inneren, orientierenden Tätigkeit oder lässt sich in solche transformieren.
2. Diese Anteile liegen als Operationen vor oder lassen sich von Handlungen in Operationen überführen (Operationalisierbarkeit).
3. Es liegt Wissen vor, mit dem sich diese Operationen (vollständig) reproduzieren lassen – reproduzierbares Wissen.
4. Dieses Wissen ist mitteilbar.
5. Es kann in öffentliches Wissen überführt werden.“ (Dahme/Raeithel 1997, S. 6)

Wir wollen uns diesem nun ein wenig nähern. Die Tätigkeitstheorie unterscheidet drei Analyseebenen einer Tätigkeit

- Tätigkeit und Motiv (**Warum** möchte ich das?)
- Handlung und Ziel (**Was** will ich mit der Handlung erreichen?)
- Operationen und Bedingungen (**Wie** kann man das erreichen?)

Da Operationen nur noch von den Bedingungen abhängen, unter denen sie stattfinden können, (und nicht mehr vom „wertenden“ Subjekt) haben sie die Potenz zur Automatisierung, d. h., sie können gegebenenfalls völlig aus einer Tätigkeit herausgenommen und einem Automaten übertragen werden. Folglich können nur Operationen in Software übertragen bzw. durch Software ersetzt werden. Dabei ist jedoch zu beachten, dass nicht alles was automatisierbar ist auch automatisierungswürdig sein muss. Hier ist ein Abwägungsprozess unter anderem im Sinne von Aufwand und Nutzen erforderlich.

Für die Gestaltung von brauchbarer Software spielen aber auch die beiden anderen Analyseebenen eine entscheidende Rolle (siehe Abbildung 1).

Analyseebene	Leitfrage	Beispiele für softwareentwicklungsrelevante Gestaltungsziele
Tätigkeit	Warum (Motiv)	Benutz- und Fehlerfreundlichkeit, Einbettung in die Anwendertätigkeit
Handlung	Was (Ziel)	Dialoggestaltung, Handhabbarkeit, visuelle Rückmeldungen, Hilfen
Operation	Wie (Verfahren)	direkt in ein Programm transformierbare, effiziente und revidierbare Objekte, Methoden und Botschaften bzw. Algorithmen und Datenstrukturen

Abbildung 1: Bedeutung der drei Analyseebenen einer Tätigkeit für die Softwareentwicklung (Dahme/Raeithel 1997, S. 7)

Andererseits erfordert die Konstruktion von Software eine Spezifikation, d. h. eine genaue Beschreibung dessen, was in Software übertragen werden sollte. Aus tätigkeitstheoretischer Sicht entspricht das der Beschreibung der in Software zu übertragenden Operationen. Das erfordert Wissen über diese Operationen. Welches Wissen über diese Operationen ist nun relevant?

„Für die Transformation in ... Software kommen in Frage:

- (1) individuell reproduzierbares Wissen (Können) und
- (2) mitteilbares individuelles Wissen (Privatwissen).

Wenn man in der Lage ist, von einer vorgegebenen Ausgangssituation zu einer vordefinierten, angestrebten Situation zu kommen, verfügt man über individuell reproduzierbares Wissen, auch kurz Können genannt. Es bezieht sich darauf, wie man das

ins Auge gefaßte Ziel einer Handlung zuverlässig erreicht. Es ermöglicht die Rekonstruktion bzw. die Wiederholung dieser Handlung und man weiß, was man tun muß, um in die Nähe der angestrebten Situation zu kommen. Mit anderen Worten, man verfügt wenigstens über ein gewohntes, flexibel einsetzbares Verfahren. Eine bewährte und selbstregulativ wirksame, intuitiv einsetzbare Methode ist dagegen schon eine höhere Stufe des Könnens, die typische Form, die das Expertenwissen annimmt. Im für die Softwareherstellung günstigsten Fall kennt man sogar einen Algorithmus, um zum angestrebten Resultat zu kommen.

Wenn wir unser individuelles Wissen anderen Menschen zur Verfügung stellen können, handelt es sich um mitteilbares Wissen ...

Mitteilbares Wissen, das nicht mehr an die Person gebunden ist, die es hervorgebracht hat, weil es in einer mitgeteilten, öffentlichen, kopierbaren und für das angesprochene Publikum verständlichen Form vorliegt, nennen wir öffentliches Wissen.“ (Dahme/Raeithel 1997, S. 10)

Entscheidend ist nun die Frage, inwieweit individuell reproduzierbares Wissen und mitteilbares individuelles Wissen über solche Operationen sich in öffentliches Wissen überführen lassen. Denn nur öffentliches (Verfahrens-) Wissen kann (transparent) in Software übertragen werden (Dahme/Raeithel 1997, S. 10).

So entstand neben dem gegenstandsorientierten Modellansatz der tätigkeitstheoretisch orientierte Ansatz. Beide Ansätze sollen nun näher eingeordnet werden.

3 Wurzeln der angewandten Informatik

Will man die wissenschaftlichen Grundlagen der Softwareentwicklung ergründen, so sind mindestens drei Richtungen oder Sichtweisen zu unterscheiden (Dahme 2008):

- A) die Sicht des **Anwenders**,
was mit dem Computer unterstützt beziehungsweise durch den Computer (wünschenswerter Weise) realisiert werden sollte,
- B) die Sicht des **Informatikers** (des Soft- oder Hardwareentwicklers),
was aus dieser Sicht mit einem Computer möglich ist oder wie Software beziehungsweise Hardware zu konstruieren ist, um eine vorgegebene

Spezifikation zu erfüllen. (Diese Sicht entspricht der so genannten „Kerninformatik“.),

C) die Sicht des „**Interface-Designers**“.

Aus dieser Sicht sollte die Interaktionsebene der Software (Mensch-Maschine-Interaktion) so gestaltet sein, dass der Anwender intuitiv (und motiviert) damit umgehen kann, d. h., diese sollte eine gute virtuelle Nachbildung der realen Situation des Anwenders darstellen, damit er sich in dieser schnell zurecht finden kann. Hier spielen Begriffe wie „usability“, brauchbare Software, Benutzerfreundlichkeit, Fehlerfreundlichkeit und dergleichen eine Rolle (siehe Dahme/Raeithel 1997, Dahme 1995).

(Im klassischen Ingenieurbereich würde man diese Richtung der industriellen Formgestaltung zuordnen.)

Aus Sicht der Softwareentwicklung könnte man A) den frühen Phasen, B) der Softwarekonstruktion und C) der Einbettung der Software in den Anwendungszusammenhang zuordnen.

Im Folgenden werde ich mich auf die Anwendersicht und damit auf die frühen Phasen der Softwareentwicklung beschränken. Versucht man der Frage nachzugehen, wo wissenschaftshistorisch der Computer bezogen auf seine Anwendung seine Wurzeln hat, so lassen sich drei identifizieren (siehe Dahme 2002, 2008):

- a) Der Computer zur Unterstützung bis hin zur Automatisierung des Rechnens ganz allgemein oder speziell: Der Computer als Mittel zur Unterstützung von wissenschaftlichen und kommerziellen Berechnungen auf der Basis mathematisierter Modelle. Letzteres werde ich als **gegenstandsorientierten Modellansatz** bezeichnen.
- b) Automatisierung von (Teilen von) Tätigkeiten durch Übertragung an einen Computer. Diese Wurzel bezeichne ich als **tätigkeitstheoretisch orientierten Ansatz**.
- c) Psychische Phänomene werden als „Informationsverarbeitung“ rekonstruiert. Dieses führt zum **kognitivistischen Ansatz** (siehe Pickert 2007).

Zu jedem dieser Ansätze gibt es eine eigene Interpretation von Software. Aus Sicht des gegenstandsorientierten Modell-Ansatzes kann Software als Automatisierung der „Berechnung“ von (mathematisierten) Modellen verstanden werden. Grundlage für die Software ist hier ein Modell des Gegenstandes.

Aus Sicht des tätigkeits-theoretisch orientierten Ansatzes kann Software als Automatisierung von operationalisierbaren Anteilen menschlicher Tätigkeit verstanden werden oder als Resultat der Transformation von Anteilen realer oder möglicher menschlicher Tätigkeit in eine maschinelle Form.

Aus Sicht des kognitivistischen Ansatzes kann Software als Automatisierung (Maschinisierung) eines Algorithmus, der wiederum selbst eine Problemlösung repräsentiert, verstanden werden. (Diese Beschreibung ist sehr eng mit dem in der theoretischen Informatik verwendeten Begriff verbunden.)

4 Der Zusammenhang zwischen diesen Wurzeln

Der kognitivistische Ansatz kann als ein spezieller gegenstandsorientierter Modellansatz der kognitiven Psychologie verstanden werden (siehe Klix 1971, Pickert 2007).

Mit Hilfe der Theorie elementarer sozialer Systeme (Dahme 1987, 1997) konnte gezeigt werden, dass ein elementares soziales System nichts anderes als eine systemtheoretische Beschreibung für eine Tätigkeit ist (Dahme 1997, S. 88), d. h., dass für elementare soziale Systeme die Tätigkeitstheorie gilt. Damit ist mit Hilfe der Theorie elementarer sozialer Systeme die Tätigkeitstheorie systemtheoretisch eingebettet. Da andererseits die Theorie elementarer sozialer Systeme die Bildung von Modellen ermöglicht, lässt sich der tätigkeitstheoretisch orientierte Ansatz auf diese Weise in einen gegenstandsorientierten Modellansatz überführen und einbetten.

Das führt zu der Aussage: Software basiert auf einem Modell, unabhängig davon, ob man sich dessen bewusst ist oder nicht.

Aus Sicht der Tätigkeitstheorie sind gegenstandsorientierte Modelle Gegenstand innerer orientierender Tätigkeit (Dahme/Raeithel 1997, S. 8-9) und sie erfüllen die Kriterien der (vollständigen) Reproduzierbarkeit und (vollständigen) Mitteilbarkeit. Damit kann die Bildung von und das Experimentieren mit Modellen tätigkeitstheoretisch eingeordnet werden. Auf diese Weise lässt sich der gegenstandsorientierte Modellansatz tätigkeitstheoretisch einbetten.

Das führt zu der Aussage: Der gegenstandsorientierte Modellansatz und der tätigkeitstheoretisch orientierte Ansatz sind gleichberechtigt.

5 Aufgabenklassen von M. M. Lehman und der tätigkeitstheoretisch orientierte Ansatz

M. M. Lehman beschäftigte sich schon seit den siebziger Jahren mit der sogenannten Softwareevolution. In diesem Zusammenhang unterschied er (M. M. Lehman 1980) drei Arten von Aufgabenklassen (bzw. Software-Typen):

- Software/Aufgaben vom S-Typ mit S für Spezifikation,
- Software/Aufgaben vom P-Typ mit P für Problem,
- Software/Aufgaben vom E-Typ mit E für (sozial) eingebettet.

Diese Aufgabenklassen bzw. Software-Typen können wie folgt charakterisiert werden:

Software/Aufgaben vom *S-Typ*

- können vollständig durch eine formale Spezifikation beschrieben werden,
- können gut nach dem Paradigma bearbeitet werden: „Erst spezifizieren – dann programmieren – dann verifizieren“,
- die Lösung einer solchen Aufgabe ist beweisbar.

Die Entwicklung ist erfolgreich, wenn die Programme der Spezifikation entsprechen.

Software/Aufgabe vom *P-Typ*

- löst ein spezifisches, abgegrenztes Problem
- wobei nicht in jedem Fall formal beschrieben werden kann, was eine Lösung ist.

Ob die gefundene Lösung annehmbar ist,

- kann nicht bewiesen,
- kann nur durch Erprobung und (qualitative) Bewertung bestätigt oder verworfen werden.

Software/Aufgaben vom *E-Typ*

(eine in der realen Welt sozial eingebettete Anwendung)

Die Anforderungen

- sind häufig nicht klar,
- müssen noch ausgehandelt werden,
- sind Änderungen unterworfen.

Die erstellten Systeme

- verändern das Arbeits- oder Erlebnisumfeld ihrer Benutzer,
- sind deshalb erhöhter Kritik ausgesetzt,
- führen in der Regel zu Rückkopplungen, (Änderungswünsche, neue oder veränderte Anforderungen) die Auslöser für weitere Entwicklungszyklen sein können.
- Die Entwicklung ist erfolgreich, wenn die Anwender mit der Software zufrieden sind.

Aus Sicht von M. M. Lehman ist nur die Software vom S-Typ stabil, d. h. der Softwareentwickler hat nur diese Situation vollständig „im Griff“. Software vom P- und E-Typ ist dagegen der Evolution unterworfen.

Aus tätigkeitstheoretischer Sicht lässt sich dieses noch besser begründen, wenn man diese Aufgabenklassen danach beurteilt, inwieweit die Aufgabe die tatsächliche Realität beschreibt, inwieweit reproduzierbares Wissen über die in Software zu transformierenden Operationen vorhanden ist und inwieweit die-

ses alles mitteilbar ist und gegebenenfalls in der Form von öffentlichem Wissen vorliegt oder sich in solches überführen lässt.

Der S-Typ

Betrachten wir zuerst die Aufgabenklasse vom S-Typ, so ist diese aus tätigkeitstheoretischer Sicht durch folgendes charakterisiert:

1. Der Teil, der in Software transformiert werden soll, ist vollständig beschrieben:
 - Es sind alle operationalisierbaren Anteile erkannt.
 - Es sind alle Operationen bestimmt, die automatisierungswürdig sind und damit die Potenz haben, in Software transformiert zu werden.
2. Reproduzierbarkeit.
 - Es gibt Wissen über die in 1. beschriebenen Operationen, mit dem diese Operationen vollständig reproduziert werden können.
3. Mitteilbarkeit.
 - Dieses Wissen ist vollständig mitteilbar und liegt in der Form von öffentlichem Wissen vor.

Darüber hinaus wird beim S-Typ noch verlangt:

4. Softwarekonstruktion.
 - Die Korrektheit der Softwarelösung ist beweisbar.

Diese Bedingungen stellen die für die Softwareentwicklung günstigste Situation dar, da sich die Spezifikation auf Grund von 1., 2. und 3. vollständig beschreiben lässt und darüber hinaus die Korrektheit der Softwarelösung beweisbar ist. Folglich hat der Softwareentwickler diese Situation vollständig „im Griff“. Nur für diesen Fall ist das Wasserfall-Modell als Vorgehensmodell geeignet.

Der P-Typ

Betrachten wir nun die Aufgabenklasse vom P-Typ, so stimmt sie bezüglich 1. „vollständig beschrieben“ und 3. „vollständig mitteilbar“ mit dem S-Typ überein. Dagegen ist die Reproduzierbarkeit beim P-Typ nicht oder nur zum Teil gegeben. Es liegt folglich eine **Reproduzierbarkeitsbarriere** vor, d. h. es gibt Wissensdefizite (und damit ein Problem) in Bezug auf die Reproduktion (das kann zum Beispiel daran liegen, dass noch keine oder keine ausreichende Theorie hierfür bekannt ist).

Aus pragmatischer Sicht versucht man in solchen Situationen das Problem nicht zu lösen, sondern mit Hilfe von Heuristiken zu umgehen (Beispiel Wetterprognosen, Baustatik).

Damit kann die Korrektheit der Softwarelösung auch nicht bewiesen werden, d. h. die Softwarelösung kann gegebenenfalls nach dem Kriterium der Genügsamkeit beurteilt werden.

Andererseits sind mit der Verwendung von Heuristiken jedoch Risiken verbunden, da mit der Anwendung von Heuristiken die Erreichung eines Zieles nicht garantiert werden kann. Daher eignet sich als Vorgehensmodell für den P-Typ das Spiralmodell von Barry Boehm, welches Risiken zu berücksichtigten versucht.

Der E-Typ

Die Aufgabenklasse vom E-Typ erweist sich jedoch als am problematischsten. Hier können sowohl Defizite bezüglich der Beschreibung als auch bezogen auf die Reproduzierbarkeit und die Mittelbarkeit auftreten. Solche Defizite sind für den E-Typ charakteristisch, insbesondere bezogen auf die Mittelbarkeit. Man kann das auch so beschreiben, dass Mittelbarkeits-, Beschreibungs- bzw. Reproduzierbarkeits-Barrieren typischer Weise vorliegen können.

Das Haupthindernis für die Gestaltung von brauchbarer Software ist hier die Mittelbarkeitsbarriere, die maßgeblich auf unterschiedliche Kulturen der Beteiligten zurückzuführen ist (Klepin 2009).

Andererseits kommt es auch häufiger vor, dass die Reproduzierbarkeit zwar gegeben ist, aber das Wissen darüber nicht in Form von öffentlichem Wissen vorliegt. Hier wird das Wissen über die Reproduzierbarkeit häufig durch Imitation kommuniziert. Solange dieses Privatwissen nur durch Imitation kommuniziert wird, ist es der Überführung in Software nicht zugänglich.

Für Aufgaben vom E-Typ eignet sich nur ein evolutionäres Vorgehen im Sinne von Prototyping (Dahme/Hesse 1997, Pomberger/Weinreich 1997), welches eine Machbarkeitsstudie voraussetzen sollte.

Fazit

Der tätigkeitstheoretisch orientierte Ansatz ermöglicht es einfacher zu erkennen, was sich erfolgreich in Software überführen lässt und welche Barrieren gegebenenfalls eine erfolgreiche Überführung behindern können, sowie welches Vorgehen in welcher Situation geeignet ist.

6 Literatur

- [1] BOEHM, B. W. (1988): A Spiral Model of Software Development and Enhancement. *IEEE Computer* 21 (5), S. 61-72.
- [2] DAHME, CH. (1987): Methodologische und theoretische Voraussetzungen für die Analyse komplexer Entscheidungssituationen. *Dissertation B, Humboldt-Universität zu Berlin, Berlin 1987.*
- [3] DAHME, CH. (1989): Ziele und Zielvorstellungen. *Deutsche Zeitschrift für Philosophie* (3), S. 263-268
- [4] DAHME, CH. (1990): Selbstorganisation und Tätigkeitstheorie. In *Niedersen, U. (Hrsg.), Selbstorganisation – Jahrbuch für Komplexität in den Natur-, Sozial und Geisteswissenschaften, Bd. 1, Selbstorganisation und Determination; Duncker & Humblot, Berlin, S. 149 ff.*
- [5] DAHME, CH. (1991): Selforganization and Activity Theory – About the Special Quality of the Building up Models of Social Objects. In *Ebeling, W.; Peschel, M. & Weidlich, W. (Hrsg.), Models of Selforganization in Complex Systems – MOSES. Volume 64 of Mathematical Research, Akademie Verlag, Berlin, S. 296-303.*
- [6] DAHME, CH. (1993): Simulation eines nichtklassischen Zuordnungsproblems dargestellt an der Kurmittelplanung eines Kliniksankatoriums. In: *Sydow, A. (Hrsg.), Simulationstechnik, 8. Symposium in Berlin, Tagungsband, Vieweg Braunschweig/Wiesbaden.*
- [7] DAHME, CH. (1995): Softwareentwicklung mit HyperCard – Benutzerfreundliche Interfacegestaltung. *Bonn: Verlag Addison-Wesley (bis Herbst 1995: Verlag Technik, Berlin).*
- [8] DAHME, CH. (1996): Zu einigen Voraussetzungen und Möglichkeiten der Synergetik in den Sozialwissenschaften. *Ethik und Sozialwissenschaften* 7 (4), S. 605-607.
- [9] DAHME, CH. (1997): Systemanalyse menschlichen Handelns – Grundlagen und Ansätze zur Modellbildung. *Opladen: Westdeutscher Verlag.*
- [10] DAHME, CH. (1998): Ein Handlungskonzept zwischen Selbstorganisation und Tätigkeitstheorie. *Ethik und Sozialwissenschaften* 9 (1), S. 23-25.
- [11] DAHME, CH. (2002). Historische, wissenschaftstheoretische und kulturelle Wurzeln der angewandten Informatik. In: *Nake, F; Rolf, A. & Siefke, D. (Hrsg.), Wozu Informatik? Theorie zwischen Ideologie, Utopie und Phantasie. Bericht 2002 - 25, TU Berlin, Fakultät IV - Elektrotechnik und Informatik, S. 52-54.*

- [12] DAHME, CH. (2008): Wissenschaftliche Grundlagen der angewandten Informatik – in memoriam Tadeusz Kasprzak. In: *Zastosowanie technologii informacyjnych do wspomagania zarządzania procesami gospodarczymi, redakcja naukowa: Nina Siemieniuk, Romuald Mosdorf. Białystok.*
- [13] DAHME, CH. & HESSE, W. (1997): Evolutionäre und kooperative Software-Entwicklung. *Informatik-Spektrum 20 (1), S. 3-4.*
- [14] DAHME, CH. & RAEITHEL, A. (1997): Ein tätigkeitstheoretischer Ansatz zur Entwicklung von brauchbarer Software. *Informatik-Spektrum. 20 (1), S. 5-12.*
- [15] ENGSTRÖM, Y. (1987): Learning by Expanding. An activity-theoretical approach to developmental research. *Helsinki: Orienta-Konsultit.*
- [16] FLOYD, CH.; KRABEL, A.; RATUSKI, S. & WETZEL, I. (1997): Zur Evolution der evolutionären Systementwicklung – Erfahrungen aus einem Krankenhausprojekt. *Informatik-Spektrum. 20 (1), S. 13-20.*
- [17] KLEPIN, M. (2009): Probleme der Mitteilbarkeit bei der Softwareentwicklung in sozial eingebetteten Systemen. *Bachelorarbeit, Humboldt-Universität zu Berlin, Institut für Informatik.*
- [18] KLIX, F. (1971): Information und Verhalten. *Berlin: Dt. Verlag der Wissenschaft. – 4. Auflage.*
- [19] LEONTJEW, A. N. (1979): Tätigkeit Bewußtsein Persönlichkeit. *Berlin: Volk und Wissen.*
- [20] LEHMAN, M.M. (1980): Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE 68 (9), S. 1060-1076.*
- [21] LEHMAN, M. M. & BELADY, L. A. (Hrsg.) (1985): Program Evolution: Processes of Software Change. *London, etc.: Academic Press.*
- [22] MÜLLER, G. (2007): Anwenderorientierte Softwarespezifikation mit der UML 2 aus tätigkeitstheoretischer Sicht. *Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik.*
- [23] POMBERGER, G. & PREE, W. (2004): Software Engineering – Architektur-Design und Prozessorientierung, 3. Auflage, *Hanser Verlag.*
- [24] POMBERGER, G. & WEINREICH, R. (1997): Qualitative und quantitative Aspekte prototypingorientierter Software-Entwicklung – Ein Erfahrungsbericht. In: *Informatik-Spektrum 20 (1), S. 33-37.*
- [25] PICKERT, G. (2007): Der kognitivistische Ansatz in der angewandten Informatik. *Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik.*

- [26] SCHULZ, L. (2009): Vorgehensmodelle in der Softwareentwicklung aus der Sicht der Aufgabenklassen von M. M. Lehman. *Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik.*
- [27] WYGOTSKI, L. S. (1964): Denken und Sprechen, *Berlin.*
- [28] WYGOTSKI, L. S. (1985): Ausgewählte Schriften, *Berlin.*